Race 01

Half Marathon C++

August 25, 2020

ucode

Contents

Engage 2

Investigate 3

Act: Basic 6

Act: Creative 8

Share 9

ucode

Engage

DESCRIPTION

Hey, coders!

Now, it's time to use all the knowledge obtained during the Half Marathon and conquer the final challenge. During this Race, you will create something truly practical.

This is an excellent opportunity to improve your threading, parsing, GUI development, and Web skills in C++.
Parsing is a complex activity, but it isn't new to you, you've encountered it in almost every project in ucode. It can be described as analysis of input with the goal of making it understandable to your program. Nowadays, it's widely used in searching patterns in Web pages to collect information and perform different actions. For example, you could use parsing to find all mentions of pink ponies on a Wiki site, or recognize and fill a captcha automatically. seconds, to write script that can fill captcha instead of you.

Another big part of this challenge is multithreading, an essential concept in modern programming. Modern computers and phones support multicore operations and one core can contain several threads, so it's very important to use all the resources you have. By researching multithreading you'll encounter numerous ways to achieve it. However, don't mistake multithreading for concurrency, these two concepts are different.

You are free to implement any multithreading algorithm as long as it works. If you want to experiment a little, try comparing execution time of your program with and without multithreading (it could even count as an extra feature, if executed well). Multiple threads communicate among each other by sharing resources, and, therefore, can speed up a program. However, it's computationally expensive to switch context among multiple threads. Because of that, you can see a degradation in your program's overall performance when using threads.

The number of internet pages is extremely large, even the largest crawlers fall short of making a complete index. For this reason, the search engines struggled to give relevant search results in the early years of the World Wide Web, before 2000. Today, relevant results are given almost instantly. This is the issue that you will be facing in this challenge.

BIG IDEA

Search engine.

ESSENTIAL QUESTION

How to process a huge number of web pages in seconds?

CHALLENGE

Create a web crawling application.

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students around you and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- What stages of IT-product creation can you name?
- How to distribute various roles in a small team and release a program?
- How to choose a graphics library that fully meets the challenge's requirements?
- What tools for parsing are available in C++?
- What tools for threading are available in C++?
- What is the best way to find text across Web pages?
- How to build a great program UX/UI?
- What features are needed to be implemented to make an interface user-friendly?
- What are the limitations during product implementation you need to deal with?
- What features make a program usable?

GUIDING ACTIVITIES

Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

- Meet with your teammate. Discuss how you will organize teamwork.
- Choose a gitflow for effective team interaction.
- Create a work plan for this challenge. Make sure you clearly understand what you need to do.
- Identify and revise the topics that you have struggled with during the Sprints. This challenge is a good opportunity to refresh and gain new knowledge and skills.
- Plan out the GUI that you want to implement in the app.
- Refresh your knowledge about parsing and threading. Glance over your previous tasks on this topic and try to get more information online.
- Clone your git repository that is issued on the challenge page in the LMS.
- Read the story, taking everyone's ideas on board.
- Distribute tasks between all team members.
- Start to develop the solution. Offer improvements. Test your code.
- Communicate with students and share information.

ANALYSIS

Analyze your findings. What conclusions have you made after completing guiding questions and activities? In addition to your thoughts and conclusions, here are some more analysis results.

- Challenge has to be carried out by the entire team.
- Each team member must understand the challenge and realization, and be able to reproduce it individually.
- It is your responsibility to assemble the whole team. Phone calls, SMS, messengers are good ways to stay in touch.
- You can proceed to `Act: Creative` only after you have completed all requirements in `Act: Basic`. But before you begin to complete the challenge, pay attention to the program's architecture. Take into account the fact that many features indicated in the `Act: Creative` require special architecture. And in order not to rewrite all the code somewhen, we recommend you initially determine what exactly you will do in the future.
- Be attentive to all statements of the story.
- Analyze all information you have collected during the preparation stages. Try to define the order of your actions.
- Perform only those tasks that are given in this document.
- Submit your files using the layout described in the story. Only useful files allowed, garbage shall not pass!
- Tasks in `shell` must be executed with `zsh`.
- Compile files with commands `cmake . -Bbuild && cmake --build ./build` that will call `CMake` and build an app.
- Pay attention to what is allowed in a certain task. Use of forbidden stuff is considered a cheat and your tasks will be failed.
- Complete tasks according to the rules specified in the Google C++ Style Guide.
  But there are several exceptions for the guide listed below:
  - you can use `#pragma once` directive instead of `#ifndef ... #define`
  - variables can be written in `mixed case`
  - class data members must begin with `m_` prefix (m for member)
  - indent 4 spaces at a time
  - each line of text in your code must be at most 120 characters long
  - ignore the sections Inputs and Outputs and Legal Notice and Author Line in the style guide
- The solution will be checked and graded by students like you. Peer-to-Peer learning.
- You must use the this project structure consisted of logical parts: the main application, own libraries and 3dparty libraries. Each individual unit must know only about its resources and, if necessary, link other libraries to itself.

```
>tree project_dir --dirsfirst --charset=ascii
project_dir
|-- app
|   |-- src
|   |   |-- CMakeLists.txt
|   |   `-- ...
```

```
|   |-- resources
|   |   `-- ...
|   |-- CMakeLists.txt
|   `-- main.cpp
|-- lib1
|   |-- CMakeLists.txt
|   `-- ...
...
|-- libN
|   |-- CMakeLists.txt
|   `-- ...
|-- 3dparty
|   |-- lib1
|   |  |-- CMakeLists.txt
|   |  `-- ...
|   ...
|   |-- libN
|   |  |-- CMakeLists.txt
|   |  `-- ...
`-- CMakeLists.txt
```

- If you have any questions or don't understand something, ask other students or just Google it.
- In the name of Talos, use your brain!

Act: Basic

NAME

Web crawler

DIRECTORY

`./`

BINARY

race01

DESCRIPTION

Create a `GUI app` that finds a fragment of text on web pages.

The app:

- searches a fragment of text at the specified link and all its sub-links
- inspects web pages in the following order (without repeating):
  1. the original given web page
  2. pages linked in that original web page
  3. pages linked in those child pages
  4. and so on
- has a graphical interface with the following features:
  - an ability for the user to set the input data parameters:
    * `URL` - the URL of the Web page where the search starts for a given fragment of the text
    * `threads count` - the number of simultaneously processed Web pages
    * `text` - the fragment of text to search
    * `limit` - the number of scanned Web pages after which the app stops searching
    * `stop if found` - a switcher that allows the user to stop the search as soon as the first occurrence of the text fragment was found or go though entire Web pages tree to find all possible results
  - showing the current status of scanning (e.g., as a log or running jobs)
  - showing, in any format, the result of scanning with every URL listed after the search was finished (e.g. summary across all URLs, or more data for every URL). For example:
    * `[URL1] - <text> was found 100500 times`
    * `[URL2] - <text> was not found`
    * `[URL3] - 404. Page Not Found`
  - `start` and `stop` buttons to manage the search

- handles all possible errors (HTTP, runtime, GUI, etc.)
- uses multithreading. Some kind of `thread pool` must be implemented to handle multiple threads. The URLs found on Web pages must be stored, picked up and processed by an idle thread

You are free to use any libraries/frameworks to implement the GUI and the network part of the app.

Act: Creative

DESCRIPTION

Listed below are a few ideas you can add to your program. But don't focus only on them you can come up with everything you want to improve your app.

- Implement a `pause` button
- Show `progress of search` using a progress bar
- Add more useful `input data` fields
- Allow users to `customize` the search and the entire app to make it personalized, individual and unique
- Visualize relations of the scanned web pages or save visualisation to an `image`
- Write `logs` to a file
- Implement a `user-friendly interface`
- Implement an appealing GUI experience
- Other creative features

Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences. During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- a text post, as a summary of your reflection
- charts, infographics or other ways to visualize your information
- a video, either of your work, or a reflection video
- an audio podcast. Record a story about your experience
- a photo report with a small post

Helpful tools:

- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends
- YouTube - upload an exciting video
- GitHub - share and describe your solution
- Telegraph - create a post that you can easily share on Telegram
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.